School of Information and Computer Technology
Sirindhorn International Institute of Technology
Thammasat University
ITS351 Database Programming Laboratory

Laboratory #11: Session Handling

Creating a PHP Login Script

Most interactive websites nowadays would require a user to log in into the website's system in order to provide a customized experience for the user. Once the user has logged in, the website will be able to provide a presentation that is tailored to the user's preferences.

A basic login system typically contains 3 components:

1. The component that allows a user to register his preferred login id and password
2. The component that allows the system to verify and authenticate the user when he subsequently logs in
3. The component that shows the user's password to his registered username if the user forgets his password

Such a system can be easily created using PHP and MySQL.

Database

Before we can create a login script, we first need to create a database to store users. We will simply need the fields "username" and "password", however you can create as many fields as you wish.

```
CREATE TABLE `user` (
      `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
      `name` VARCHAR( 50 ) NOT NULL ,
      `email` VARCHAR( 50 ) NOT NULL ,
      `username` VARCHAR( 50 ) NOT NULL ,
      `passwd` VARCHAR( 50 ) NOT NULL ,
      `disable` INT NOT NULL
) ;
```

This will create a database called "user" with 6 fields: ID, name, email, username, password, and disable.

Component 1 – Registration

Component 1 is typically implemented using a simple HTML form that contains 4 fields and 2 buttons:

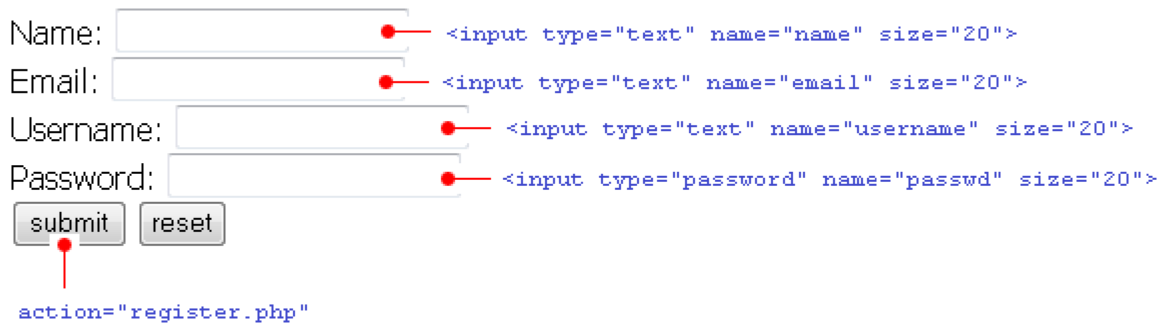
1. A preferred name field
2. A valid email address field
3. A preferred login id field
4. A preferred password field

5. A Submit button
6. A Reset button

Assume that such a form is coded into a file named signup.html. The following HTML code is a typical example. When the user has filled in all the fields, the register.php page is called when the user clicks on the Submit button.

signup.html
```
<html>
<body>
<form name="register" method="post" action="register.php">
 Name: <input type="text" name="name" size="20"><br>
 Email: <input type="text" name="email" size="20"><br>
 Username: <input type="text" name="username" size="20"><br>
 Password: <input type="password" name="passwd" size="20"><br>
 <input type="submit" name="submit" value="submit">
 <input type="reset" name="reset" value="reset">
</form>
</body>
</html>
```

Output will be


Name:
<input type="text" name="name" size="20">
Email:
<input type="text" name="email" size="20">
Username:
<input type="text" name="username" size="20">
Password:
<input type="password" name="passwd" size="20">
submit
reset
action="register.php"

The following code can be used as part of register.php to process the registration. It connects to the MySQL database and inserts a line of data into the table used to store the registration information.

register.php
```
<?php
$name = $_POST['name'];
$email = $_POST['email'];
$username = $_POST['username'];
$passwd = $_POST['passwd'];
require_once('connect.php');

$q="INSERT INTO user (name,email,username,passwd,disable)
VALUES ('".$name."','".$email."','".$username."','".$passwd."',0)";

$result = $mysqli->query($q);
if (!$result)    {
     die('Error: '.$q." ". $mysqli->error);
}
header("Location: login.html");//redirect
?>
```

The example code assumes that the MySQL table that is used to store the registration data is named "user" and contains 6 fields – the name, email, username, password, and disable fields including id, which is auto increment. The values of the $name, $email, $username, and $password variables are passed in from the form in signup.html using the post method. The default value of the disable field is set to false.

Component 2 – Verification and Authentication

A registered user will want to log in to the system to access the functionality provided by the website. The user will have to provide his login id and password for the system to verify and authenticate.

This is typically done through a simple HTML form. This HTML form typically contains 2 fields and 2 buttons:

1. A login id field
2. A password field
3. A Submit button
4. A Reset button

Assume that such a form is coded into a file named login.html. The following HTML code is a typical example. When the user has filled in all the fields, the "checklogin.php" page is called when the user clicks on the Submit button.

login.html

```
<html>
<body>
<form name="login" method="post" action="checklogin.php">
 Username: <input type="text" name="username" size="20"><br>
 Password: <input type="password" name="passwd" size="20"><br>
 <input type="submit" name="submit" value="submit">
 <input type="reset" name="reset" value="reset">
<form>
</body>
</html>
```

The following code can be used as part of checklogin.php to process the login request. It connects to the MySQL database and queries the table used to store the registration information.

checklogin.php

```
<?php
$username = $_POST['username'];
$passwd = $_POST['passwd'];

require_once('connect.php');
$q="select * from user where disable = 0
and username='".$username."' and passwd='".$passwd."'" ;

$result = $mysqli->query($q);
if (!$result)    {
  die('Error: '.$q." ". $mysqli->error);
}
$count = $result->num_rows;
```

```
// If result matches, there must be one row returned
if($count==1){
      echo "Login Sucessfully";
} else {
      echo "Wrong Username or Password";
}
?>
```

As in component 1, the code assumes that the MySQL table that is used to store the registration data is named “user” and contains 2 fields – the username and password fields. The values of the **$username** and **$passwd** variables are passed in from the form in **checklogin.php** using the post method.

For security reason, in practice all the input values from the form need to be wrapped with `$mysqli->real_escape_string()`. Hence, in the previous example, the query should be

```
"select * from user where disable = 0
and username='".$mysqli->real_escape_string($username)."' and
passwd='".$mysqli->real_escape_string($passwd)."'" ;
```

This is done to prevent a security attack known as “SQL injection” where the $username, and $passwd are input in such a way that SQL query is completed and is always evaluated to true. This allows a malicious hacker to login as anyone. For simplicity, in this lab, we will omit `$mysqli->real_escape_string()`. However, keep in mind that it is very important to use one in practice.

Component 3 – Forgot Password

A registered user may forget his password to log into the website’s system. In this case, the user will need to supply his username for the system to retrieve his password.

This is typically done through a simple HTML form. This HTML form typically contains 1 field and 2 buttons:

1. A username field
2. A Submit button
3. A Reset button

Assume that such a form is coded into a file named **forgot.html**. The following HTML code is a typical example. When the user has filled in all the fields, the **checkpass.php** page is called when the user clicks on the Submit button.

forgot.html

```
<!DOCTYPE html>
<html>
<body>
<form name="login" method="post" action="checkpass.php">
 Username: <input type="text" name="username" size="20"><br>
 <input type="submit" name="submit" value="submit">
 <input type="reset" name="reset" value="reset">
</form>
</body>
</html>
```

The following code can be used as part of checkpass.php to process the login request. It connects to the MySQL database and queries the table used to store the registration information.

checkpass.php

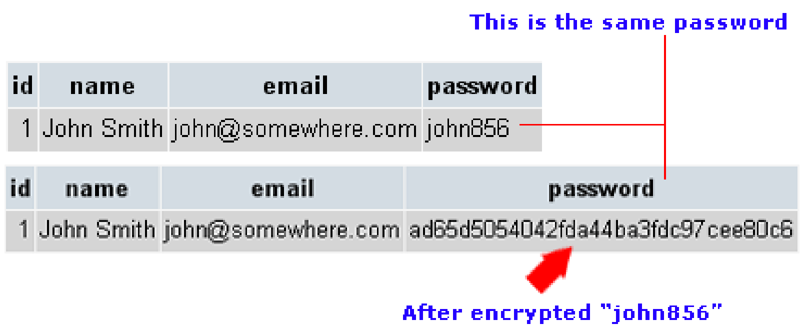
```
<?php
$username = $_POST['username'];
require_once('connect.php');

$q="select * from user where username='".$username."'" ;

$result = $mysqli->query($q);
if (!$result)    {
  die('Error: '.$q." ". $mysqli->error );
}
$count = $result->num_rows;

if($count==1){
      $row = $result->fetch_array();
      $password = $row["passwd"];
      echo "your password is ".$password;
}
else {
      echo "no such username in the system. please try again.";
}
?>
```

As in component 1, the code assumes that the MySQL table that is used to store the registration data is named "user" and contains 2 fields – the username and password fields. The value of the $username variable is passed from the form in forgot.html using the post method.

An extra caution about security to keep in mind is that, in practice when the password of a user is queried, the password is not directly shown onto the screen. Doing it this way allows anyone to see each other's password. Typically the password is sent to the registered email of the requested user. So, only those who can see the email (i.e. presumably the user himself) can see the password of this system. For simplicity, again, we directly show the password onto the screen.

Hashing Password - Make Your Password Storage More Secure

The previous example is to illustrate how a very basic login system can be implemented. In this example, we enhanced it to include password hash using md5() function. The value in password field will change as shown in following figure.



id	name	email	password
1	John Smith	john@somewhere.com	john856

id	name	email	password
1	John Smith	john@somewhere.com	ad65d5054042fda44ba3fdc97cee80c6

Look at these two databases; it's the same person and same information. In the first example, raw password is stored. In the second example, the md5 digested or hash value of the password is stored instead.

Syntax

```
<?php
      $password="123456";
      md5($password);
      // Use md5(); to hash password
?>
```

When you hash "john856" using this code, you'll see this result "ad65d5054042fda44ba3fdc97cee80c6". This is not a random result. Every time you hash the same password you will get the same result.

To hash a password, we need to do it before inserting into the database in register process. Therefore, we need to modify register.php as followed

register.php

```
<?php
$name = $_POST['name'];
$email = $_POST['email'];
$username = $_POST['username'];
$passwd = $_POST['passwd'];

require_once('connect.php');

// hash password
$passwd=md5($passwd);

$q="INSERT INTO user (name,email,username,passwd,disable)
VALUES ('".$name."','".$email."','".$username."','".$passwd."',0)";

$result = $mysqli->query($q);
if (!$result)    {
      die('Error: '.$q." ". $mysqli->error);
}

header("Location: login.html"); //redirect
?>
```

In this example, we put a record with username and password into table. This password was hashed by md5() function from the real password.

To check login, you need to hash the input password using md5() function and match it with the already hashed password in the “user” table. Therefore, we need to modify checklogin.php as followed

checklogin.php

```
<?php
$username = $_POST['username'];
$passwd = $_POST['passwd'];

require_once('connect.php');
// hash password
$passwd=md5($passwd);
```

```
$q="select * from user where disable = 0
and username='".$username."' and passwd='".$passwd."'" ;

$result = $mysqli->query($q);
if (!$result)    {
  die('Error: '.$q." ". $mysqli->error);
}
$count = $result->num_rows;
// If result matches, there must be one row returned
if($count==1){
      echo "Login Sucessfully";
} else {
      echo "Wrong Username or Password";
}
?>
```

md5 is a one-way hash, which means you can only hash a password to get a hash value. But, you cannot get the original password from a hash value. One advantage of storing a hash instead of the raw password is that, nobody can know the real passwords if the database is attacked (even the database administrator).

From security point of view, using md5 is not quite sufficient these days. Because of its fast nature, the system is still subject to a brute force attack using an md5 hashed dictionary. In practice, there are many more secure hashing algorithms to use. People also store what is called “salted password hash” not just password hashes. This significantly makes brute force using hashed dictionary much harder. Please learn more by yourself. For the demonstration purpose, md5 is sufficient in this lab.

Come back to the system we are developing. Since we now do not know the real password anymore, if a user forgot their password, we cannot give them their password. So the only solution is to write a script for them to **reset the password**. Therefore, we need to modify checkpass.php as followed

checkpass.php

```
<?php
$username = $_POST['username'];
require_once('connect.php');

$q="select * from user where username='".$username."'" ;

$result = $mysqli->query($q);
if (!$result)    {
  die('Error: '.$q." ". $mysqli->error );
}
$count = $result->num_rows;

if($count==1){
      $passwd = "1234";
      $newpasswd = md5($passwd);
      $q="update user set passwd ='".$newpasswd."'
      where username='".$username."'" ;
      $mysqli->query($q) or die('updating password failed: ' .
$mysqli->error);
      echo "Your password is reset to $passwd";
} else {
      echo "no such username in the system. please try again.";
}
?>
```

PHP Session Variables

Session support in PHP consists of a way to preserve certain data across subsequent accesses. A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user leaves the website. If you need a permanent storage you may want to store the data in a database.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?php session_start(); ?>

<html>
<body>
…
</body>
</html>
```

This tiny piece of code will register the user's session with the server, allow you to start saving user information and assign a UID (unique identification number) for that user's session.

Storing a Session Variable

When you want to store user data in a session, you must use the $_SESSION associative array. This is where you both store and retrieve session data. Now, as an example, let’s store user's name and their email in a session. To do so you would do the following.

example1.php

```
<?php
session_start();
$_SESSION["username"] = "john";
$_SESSION["email"] = "jh@mail.com";
?>
```

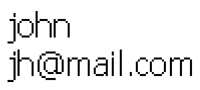
Retrieving Stored Session Information

Retrieving stored session information is really easy. You can access the stored session information on any page without doing anything extra.

example2.php

```php
<?php
      session_start();

      echo $_SESSION["username"];
      echo "<br>";
      echo $_SESSION["email"];
?>
```

Output is



```
john
jh@mail.com
```

Destroying/deleting session information

Session data is usually stored after your script terminated. If you wish to clear a session variable yourself, you can simply use the unset() function to clean the session variable.

```php
<?php
      session_start();
      unset($_SESSION["username"]);
      unset($_SESSION["email"]);
?>
```

To completely destroy all session variables in one go, you can use the session_destroy() function. Note that session_start() is needed before calling session_destroy().

```php
<?php
      session_start();
      session_destroy();
?>
```

Setting a Session to Login

In the previous code, the part that checked if the user's login is correct was this:

```php
if($count==1){
      echo "Login Sucessfully";
} else {
      echo "Wrong Username or Password";
}
```

If login information is correct, then we need to use sessions to store the user's information so they can access member-only content.

checklogin.php

```
<?php
session_start();
$username = $_POST['username'];
$passwd = $_POST['passwd'];

require_once('connect.php');
// hash password
$passwd=md5($passwd);
$q="select * from user where disable = 0
and username='".$username."' and passwd='".$passwd."'" ;

$result = $mysqli->query($q);
if (!$result)    {
  die('Error: '.$q." ". $mysqli->error);
}
$count = $result->num_rows;
// If result matches, there must be one row returned
if($count==1){
      echo "Login Sucessfully";
      // Store User Information to Session
      $row = $result->fetch_array();
      $_SESSION['id'] = $row["id"];
      $_SESSION['name'] = $row["name"];
} else {
      echo "Wrong Username or Password";
}
?>
```

Now that the user has logged in successfully, and has his id and his name stored in session variables, we can start working with member-only content. **One important thing to remember is that any page which accesses session data must have session_start(); declared at the top**.

member.php

```
<?php
session_start();

if (!isset($_SESSION['id']))
{
    // User not logged in, redirect to login page
    header("Location: login.html");
}

// Member only content
// Display Member information
echo "<p>User ID: " . $_SESSION['id'];
echo "<p>Name: " . $_SESSION['name'];

// Display logout link
echo "<br><a href=\"logout.php\">Click here to logout!</a>";
?>
```

Now only person who has authentication will be able to view this page. As mentioned above, the presence or absence of `ID` in the session will tell us whether the user is logged in or not. If a variable names `ID` exists in the session, then the user has been logged in and authenticated.

To logout the user, simply unset ID variable in session or destroy all of the session variables. See the logout.php script for example.

logout.php

```
<?php
session_start();

session_destroy();
// Logged out, return home.
header("Location: login.html");
?>
```

Worksheet

Please create simple login system with php + mysql script that includes the following files:

1. **Sigup Page:** allow user to register his information including name, email, username, and password. After complete, redirect to login page.
2. **Login Page:** allow user to login to the system. You must check whether login is successful or not. If login successfully, redirect to member page.
3. **Member Page:** allow only authenticated user to access this page. You must display all information of login user, except password, in this page and allow user to logout the system.
4. **Logout Page:** allow user to logout the system and redirect to login page.

Step of creating simple login system

1. Create table "user" in database "hr".
2. Create file signup.html
3. Create file register.php
4. Create file login.html
5. Create file checklogin.php.
6. Create file member.php
7. Create file logout.php

Exercise

Import database named “STAFF” from given resource file in database folder. The following figure shows the structure of STAFF database. Note that all fields must set to allow NULL value except primary key.

USERGROUP Table

Field	Type	Length Values	Extra	Primary Key
USERGROUP_ID	INT		Auto_increment	Yes
USERGROUP_CODE	VARCHAR	50		
USERGROUP_NAME	VARCHAR	50		
USERGROUP_REMARK	VARCHAR	255		
USERGROUP_URL	VARCHAR	50		

USER Table

Field	Type	Length Values	Extra	Primary Key
USER_ID	INT		Auto_increment	Yes
USER_TITLE	INT			
USER_FNAME	VARCHAR	50		
USER_LNAME	VARCHAR	50		
USER_GENDER	INT			
USER_EMAIL	VARCHAR	50		
USER_NAME	VARCHAR	25		
USER_PASSWD	VARCHAR	25		
USER_GROUPID	INT			
DISABLE	INT			

From the given homepage for user management project with HTML and CSS Styles, please create the login page to authenticate user before accessing any page in the project. Don't forget to handle the "forget password" case. After login, please show welcome message with full name (USER_FNAME and USER_LNAME) together with USERGROUP_NAME of login user on the top of menu in each page.

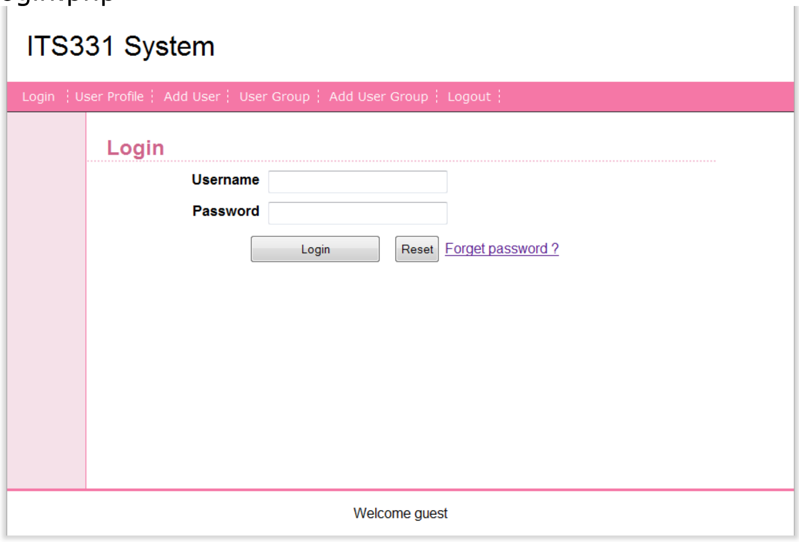
Note that user who is not disabled should be able to access pages depending on user group permission as shown in the following table:

Page Name	**Group: Admin**	**Group: Staff**	**Group: Member**
add_group.php	Yes	Yes	No
group.php	Yes	Yes	No
edit_group.php	Yes	Yes	No
del_group.php	Yes	Yes	No
add_user.php	Yes	No	Yes
user.php	Yes	No	Yes
edit_user.php	Yes	No	Yes
del_user.php	Yes	No	Yes

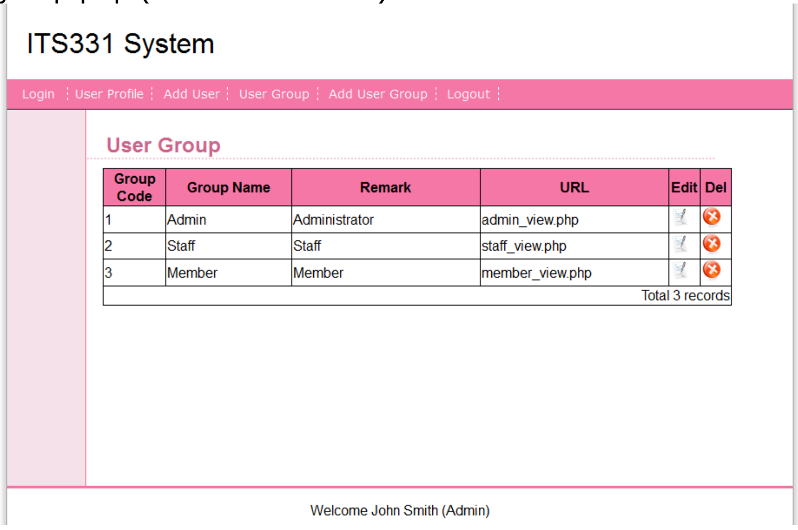
Additionally, add logout page to the project in order to allow user log out from the system.

Example of Website 1

login.php

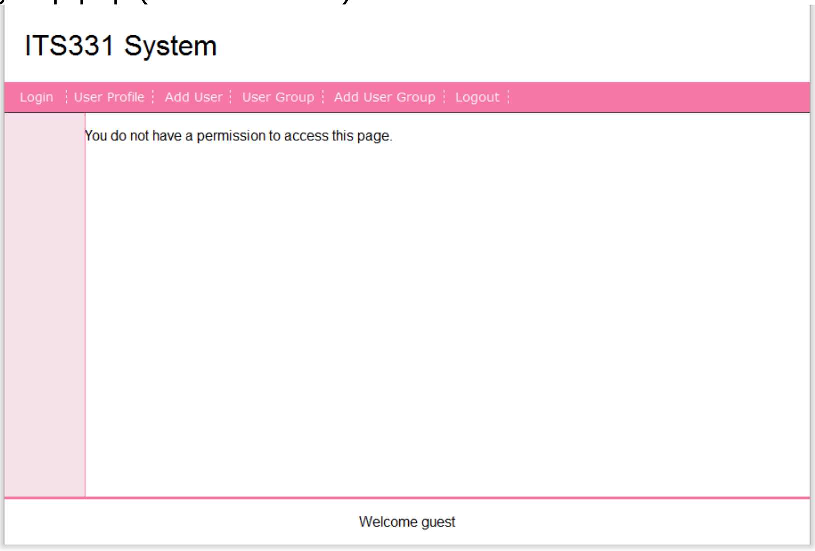


group.php (Allow Permission)


ITS331 System
Login ¦ User Profile ¦ Add User ¦ User Group ¦ Add User Group ¦ Logout
User Group
Welcome John Smith (Admin)

Group Code	Group Name	Remark	URL	Edit	Del
1	Admin	Administrator	admin_view.php		
2	Staff	Staff	staff_view.php		
3	Member	Member	member_view.php		
				Total 3 records	

group.php (No Permission)

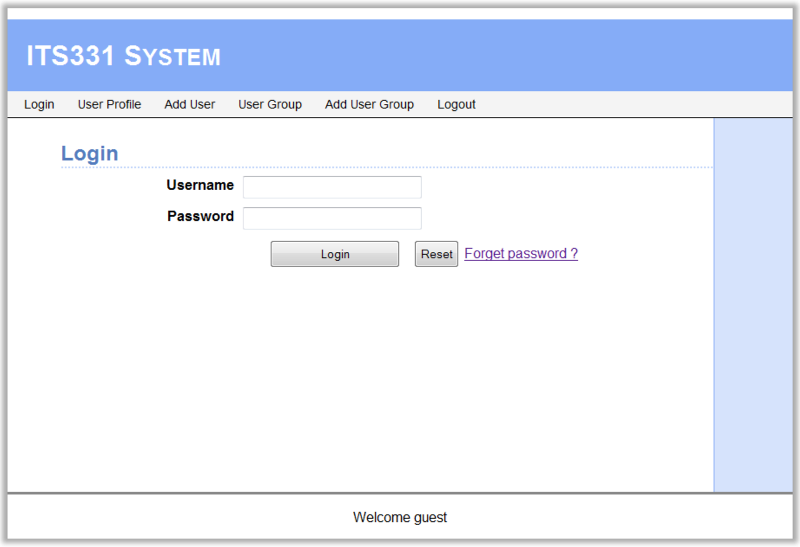

ITS331 System
Login | User Profile | Add User | User Group | Add User Group | Logout |

You do not have a permission to access this page.

Welcome guest

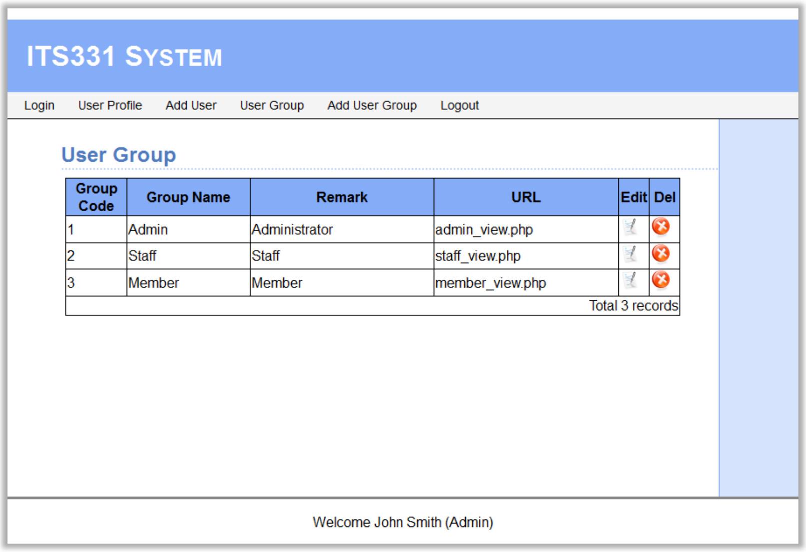
Example of Website 2

login.php



Welcome guest

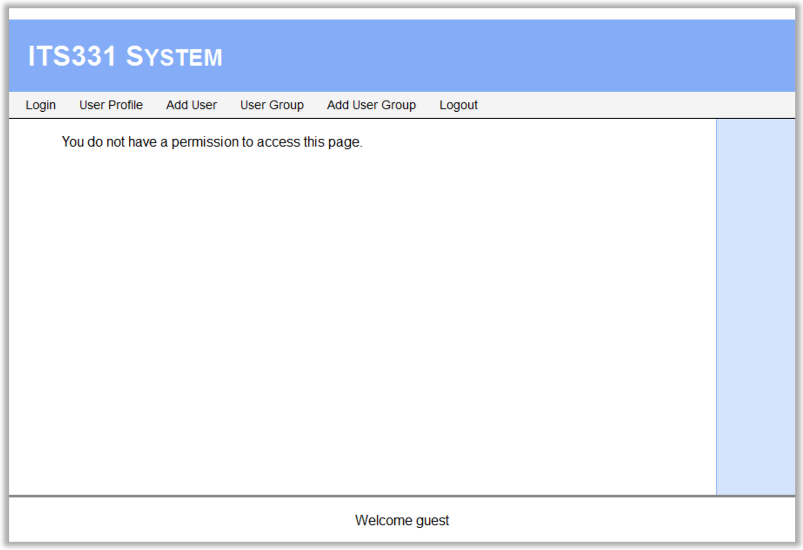
group.php (Allow Permission)


ITS331 SYSTEM
Login User Profile Add User User Group Add User Group Logout
User Group

Group Code	Group Name	Remark	URL	Edit	Del
1	Admin	Administrator	admin_view.php		
2	Staff	Staff	staff_view.php		
3	Member	Member	member_view.php		
					Total 3 records

Welcome John Smith (Admin)

group.php (No Permission)

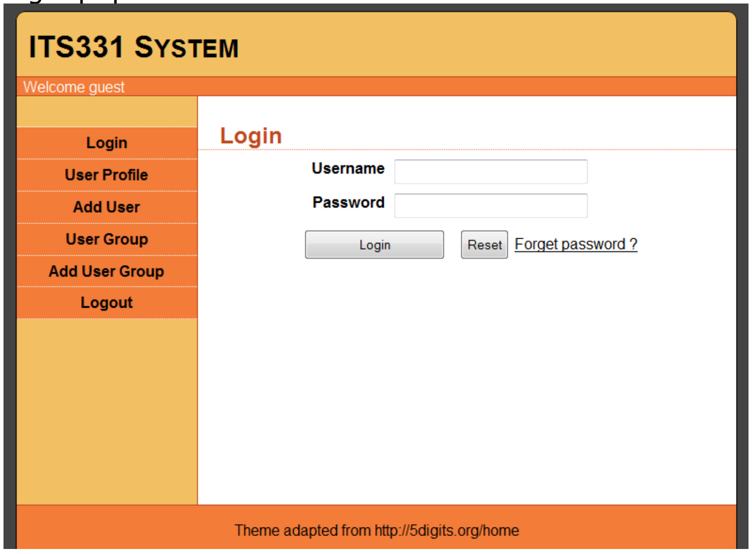


You do not have a permission to access this page.

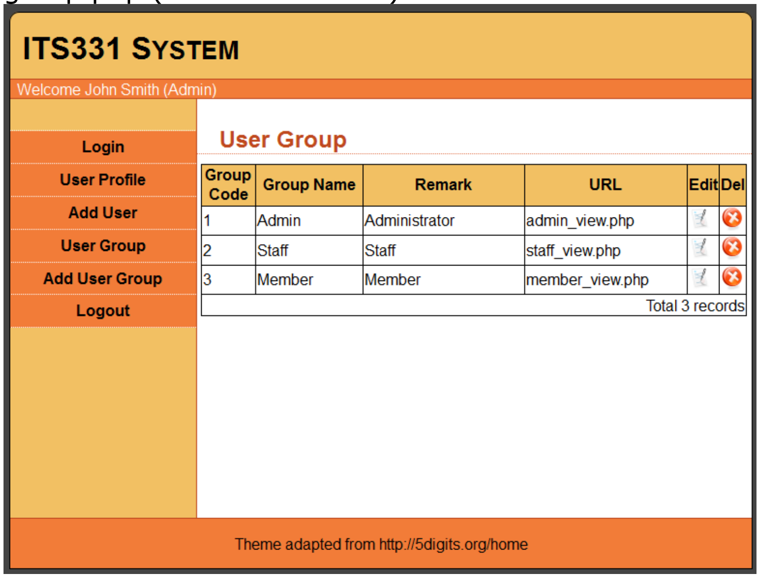
Welcome guest

Example of Website 3

login.php



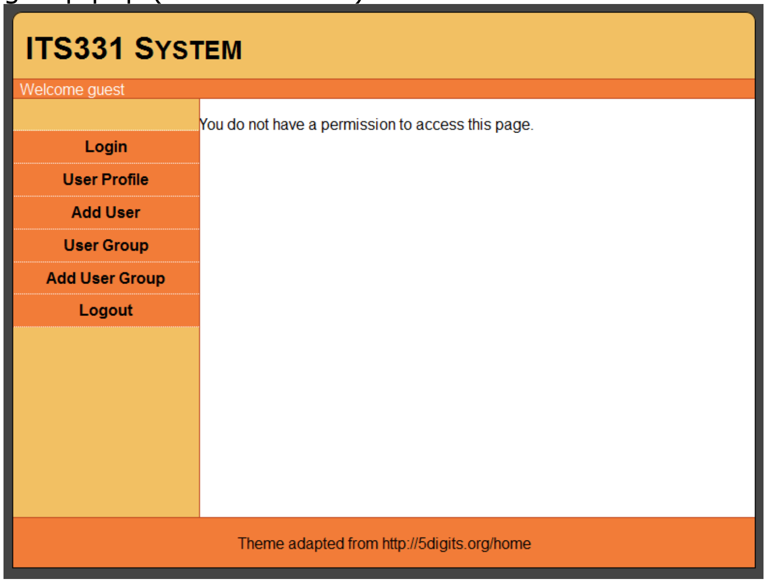
group.php (Allow Permission)



Group Code	Group Name	Remark	URL	Edit	Del
1	Admin	Administrator	admin_view.php	✎	✖
2	Staff	Staff	staff_view.php	✎	✖
3	Member	Member	member_view.php	✎	✖
					Total 3 records

Theme adapted from http://5digits.org/home

group.php (No Permission)



You do not have a permission to access this page.